对象存储 zos-sdk-go 使用手册

环境依赖

Golang 版本1.9及以上，可访问 Golang 官网下载页面进行下载。

需配置 `go env` 中的 `GO111MODULE` 选项为 `on`。

项目配置

项目初始化

下面过程以"列出桶"功能为例，说明如何从零开始创建一个 zos-sdk-go 项目。

首先在项目目录依次运行下列 go 命令初始化项目：

```
# 生成 go.mod 文件
go mod init zos-sdk-go-example

# 配置本地依赖
go mod edit -require="github.com/aws/aws-sdk-go@v1.38.63"
go mod edit -require="github.com/jmespath/go-jmespath@v0.4.0"
go mod edit -replace="github.com/aws/aws-sdk-go@v1.38.63"="{path of sdk}/aws-sdk-go-1.38.63"
go mod edit -replace="github.com/jmespath/go-jmespath@v0.4.0"="{path of sdk}/go-jmespath-0.4.0"
```

配置后的 `go.mod` 文件内容类似如下示例：

```
module zos-sdk-go-example

go 1.22.1

require (
    github.com/aws/aws-sdk-go v1.38.63
    github.com/jmespath/go-jmespath v0.4.0
)

replace github.com/aws/aws-sdk-go v1.38.63 => {path of sdk}/aws-sdk-go-1.38.63

replace github.com/jmespath/go-jmespath v0.4.0 => {path of sdk}/go-jmespath-0.4.0
```

获取访问凭证

`AccessKey`，`SecretAccessKey` 和 `Endpoint` 是调用ZOS服务必要的三个参数，具体获取方式如下：

- `AccessKey` 和 `SecretAccessKey`：可在控制台查看或通过 `OpenAPI` 的 get-keys 接口查询
- `Endpoint`：可在控制台查看或通过 `OpenAPI` 的 get-endpoint 接口查询

项目功能验证

在项目目录下创建 `main.go` 文件，内容如下：

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

var AccessKey string = "Your AccessKey"
var SecretAccessKey string = "Your SecretAccessKey"
var Endpoint string = "Your Endpoint"

func main() {
    svc := BuildClient()
    // 验证 ListBuckets 功能
    result, err := svc.ListBuckets(&s3.ListBucketsInput{})
    if err != nil {
        fmt.Printf("Unable to list buckets. %v\n", err)
    } else {
        fmt.Println(result)
    }
}

// 创建并返回一个 service client
func BuildClient() *s3.S3 {
    conf := &aws.Config{
        Endpoint:         aws.String(Endpoint),
        S3ForcePathStyle: aws.Bool(false),
        Credentials:      credentials.NewStaticCredentials(AccessKey,
SecretAccessKey, ""),
        Region:           aws.String("us-east-1"),
    }
    sess := session.Must(session.NewSessionWithOptions(session.Options{Config:
*conf}))
    svc := s3.New(sess)
    return svc
}
```

完成上述操作之后，可执行 `go run main.go` 查看运行结果，响应结果应类似：

```
{
  Buckets: [{
      CreationDate: 2024-01-01 05:59:29.634 +0000 UTC,
      Name: "exampleBucket"
    },{
      CreationDate: 2024-01-02 01:56:00.83 +0000 UTC,
      Name: "exampleBucket2"
    }],
  Owner: {
    DisplayName: "张三",
    ID: "00000000-0000-0000-0000-000000000000"
  }
}
```

桶基础操作

创建桶

功能介绍

在指定账号下创建一个新的桶。

方法原型

```
func (c *S3) CreateBucket(input *CreateBucketInput) (*CreateBucketOutput, error)
```

输入参数

参数	类型	说明	是否必要
ACL	*string	配置创建桶预定义的标准ACL信息，例如 `private`、`public-read` 等	否
Bucket	*string	创建桶的名称	是
AZPolicy	*string	桶的数据冗余策略，可选值为 `single-az` 和 `multi-az`，分别表示单 AZ 存储和多 AZ 存储。默认为单 AZ 存储。	否
StorageClass	*string	桶的存储类型，可选值为 `STANDARD`、`STANDARD_IA` 和 `GLACIER`，分别表示标准、低频、归档。默认为标准存储。	否

代码示例

```
func CreateBucket(svc *s3.S3, bucket string) {
    input := &s3.CreateBucketInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.CreateBucket(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取桶列表

功能介绍

查询请求用户拥有的所有桶的列表。

方法原型

```
func (c *S3) ListBuckets(input *ListBucketsInput) (*ListBucketsOutput, error)
```

输出结果

属性名	类型	说明
Buckets	[]*Bucket	桶信息数组，包含了每个桶的名字与创建时间
Owner	*Owner	桶的拥有者信息

代码示例

```
func ListBuckets(svc *s3.S3) {
    // ListBuckets 接口无实际参数，填空即可
    input := &s3.ListBucketsInput{}
    output, err := svc.ListBuckets(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取桶信息

功能介绍

查询用户账号下某个桶是否存在并且用户是否有权访问。

方法原型

```
func (c *S3) HeadBucket(input *HeadBucketInput) (*HeadBucketOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

代码示例

```
func HeadBucket(svc *s3.S3, bucket string) {
    input := &s3.HeadBucketInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.HeadBucket(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

删除桶

功能介绍

删除指定桶。删除一个桶前，需要先删除该桶中的全部对象。

方法原型

```
func (c *S3) DeleteBucket(input *DeleteBucketInput) (*DeleteBucketOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

代码示例

```
func DeleteBucket(svc *s3.S3, bucket string) {
    input := &s3.DeleteBucketInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.DeleteBucket(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取桶访问权限

功能介绍

获取桶的 ACL，用户在获取桶的 ACL 之前需要具备 `READ_ACP` 权限。

方法原型

```
func (c *S3) GetBucketAcl(input *GetBucketAclInput) (*GetBucketAclOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Grants	[]*Grant	授权信息，包含了每项授权和被授权人的信息
Owner	*Owner	桶的拥有者信息

代码示例

```
func GetBucketAcl(svc *s3.S3, bucket string) {
    input := &s3.GetBucketAclInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.GetBucketAcl(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

设置桶访问权限

功能介绍

设置桶的 ACL，用户在设置桶的 ACL 之前需要具备 `WRITE_ACP` 权限。

方法原型

```
func (c *S3) PutBucketAcl(input *PutBucketAclInput) (*PutBucketAclOutput, error)
```

输入参数

参数	类型	说明	是否必要
ACL	*string	配置创建桶预定义的标准 ACL 信息，可选值为 `private` 和 `public-read`	否
AccessControlPolicy	*AccessControlPolicy	配置该桶对于每个用户的 ACL 授权信息	否
Bucket	*string	桶的名称	是

`AccessControlPolicy` 的内部结构如下：

参数	类型	说明	是否必要
Grants	[]*Grant	授权信息	是
Owner	*Owner	所有者	是

`Grants` 的内部结构如下：

参数	类型	说明	是否必要
Grantee	*Grantee	被授权者	是
Permission	*string	授予权限，可选值为 `READ`，`READ_ACP`、`WRITE`、`WRITE_ACP` 和 `FULL_CONTROL`	是

`Owner` 的内部结构如下：

参数	类型	说明	是否必要
DisplayName	*string	所有者的用户名称	是
ID	*string	所有者的用户 ID	是

`Grantee` 的内部结构如下：

参数	类型	说明	是否必要
DisplayName	*string	用户名称	否
EmailAddress	*string	用户邮箱	否
Type	*string	用户类型，可选值为 `CanonicalUser`、`AmazonCustomerByEmail` 和 `Group`。`CanonicalUser` 则 `ID` 为必填项；`AmazonCustomerByEmail` 则 `EmailAddress` 为必填项，并且将会保存其指向到的 `CanonicalUser` 类型的用户；`Group` 则 `URI` 为必填项。	是
ID	*string	标准用户 ID	否
URI	*string	授权组的 URI	否

代码示例

```
func PutBucketAcl(svc *s3.S3, bucket string) {
    input := &s3.PutBucketAclInput{
        Bucket: aws.String(bucket),
        ACL:    aws.String("private"),
    }
    output, err := svc.PutBucketAcl(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取桶策略

功能介绍

获取桶的桶策略。

方法原型

```
func (c *S3) GetBucketAcl(input *GetBucketAclInput) (*GetBucketAclOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Policy	*string	JSON 格式的桶策略信息

代码示例

```
func GetBucketPolicy(svc *s3.S3, bucket string) {
    input := &s3.GetBucketPolicyInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.GetBucketPolicy(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

设置桶策略

功能介绍

设置桶的桶策略。描述桶策略的信息以 JSON 格式的字符串通过 `Policy` 参数传入。

方法原型

```
func (c *S3) PutBucketPolicy(input *PutBucketPolicyInput)
(*PutBucketPolicyOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Policy	*string	JSON 格式的桶策略信息	是

`Policy` 的内部结构如下：

参数	类型	说明
Version	*string	当前支持 "2012-10-17"
Id	*string	桶策略 Id
Statement	list	桶策略描述，定义完整的权限控制。每条桶策略的 `Statement` 可由多条描述组成，每条描述是一个 dict

`Statement` 的内部结构如下：

参数	类型	说明
Sid	*string	本条桶策略描述的ID
Effect	*string	桶策略的效果，即指定本条桶策略描述的权限是接受请求还是拒绝请求。 接受请求：`"Allow"`， 拒绝请求：`"Deny"`
Principal	dict	指定本条桶策略描述所作用的用户。以字典形式表示，支持直接使用通配符 `"*"` 表示所有用户。当对特定用户进行授权时，格式为 `{"AWS": "arn:aws:s3:::userId"}`
Action	list	指定本条桶策略描述所作用的操作。以列表形式表示，支持直接使用通配符 `"*"` 表示该资源能进行的所有操作。常用的Action有 `"s3:GetObject"`，`"s3:PutObject"`等
Resource	list	此条策略所作用的资源，如桶、对象等
Conditon	dict	条件语句，指定本条桶策略所限制的条件。可以用于对资源配置各种策略，比如设置防盗链等

代码示例

```
func PutBucketPolicy(svc *s3.S3, bucket string) {
    // 一个允许任意用户读取桶内对象的桶策略
    policy := `
    {
        "Version": "2012-10-17",
        "Id": "policy id",
        "Statement": [
            {
                "Sid": "statement id",
                "Effect": "Allow",
                "Principal": "*",
                "Action": "*",
                "Resource": [
                    "arn:aws:s3:::YOUR-BUCKET-NAME/*"
                ]
            }
        ]
    }`
    input := &s3.PutBucketPolicyInput{
        Bucket: aws.String(bucket),
        Policy: aws.String(policy),
    }
    output, err := svc.PutBucketPolicy(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

删除桶策略

功能介绍

删除桶的桶策略。

方法原型

```
func (c *S3) DeleteBucketPolicy(input *DeleteBucketPolicyInput)
(*DeleteBucketPolicyOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

代码示例

```
func DeleteBucketPolicy(svc *s3.S3, bucket string) {
    input := &s3.DeleteBucketPolicyInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.DeleteBucketPolicy(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取版本控制信息

功能介绍

`GetBucketVersioning` 操作用于获取桶的版本控制状态信息，只有桶的拥有者才能获取到桶的版本控制信息。

桶的版本控制有三种状态：未开启/开启（Enabled）/暂停（Suspended）。桶在被设置版本状态前默认为未开启状态，执行 `GetBucketVersioning` 操作将无法获得任何信息。

方法原型

```
func (c *S3) GetBucketVersioning(input *GetBucketVersioningInput)
(*GetBucketVersioningOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Status	*string	桶的版本控制设置状态，可选值为 `Enabled` 和 `Suspended`

代码示例

```
func GetBucketVersioning(svc *s3.S3, bucket string) {
    input := &s3.GetBucketVersioningInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.GetBucketVersioning(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

设置版本控制

功能介绍

设置桶的版本控制状态。

方法原型

```
func (c *S3) PutBucketVersioning(input *PutBucketVersioningInput)
(*PutBucketVersioningOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
VersioningConfiguration	*VersioningConfiguration	设置版本控制状态的参数	是

`VersioningConfiguration` 的内部结构如下：

参数	类型	说明	是否必要
Status	*string	桶的版本控制设置状态，可选值为 `Enabled` 和 `Suspended`	是

代码示例

```
func PutBucketVersioning(svc *s3.S3, bucket string) {
    input := &s3.PutBucketVersioningInput{
        Bucket: aws.String(bucket),
        VersioningConfiguration: &s3.VersioningConfiguration{
```

```
            Status: aws.String("Enabled"),
        },
    }
    output, err := svc.PutBucketVersioning(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

查看桶生命周期规则

功能介绍

查看指定桶的生命周期规则。

方法原型

```
func (c *S3) GetBucketLifecycleConfiguration(input
*GetBucketLifecycleConfigurationInput) (*GetBucketLifecycleConfigurationOutput,
error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Rules	[]*LifecycleRule	封装生命周期的数组

代码示例

```
func GetBucketLifecycleConfiguration(svc *s3.S3, bucket string) {
    input := &s3.GetBucketLifecycleConfigurationInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.GetBucketLifecycleConfiguration(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

设置桶生命周期规则

功能介绍

为指定桶设置生命周期规则。

方法原型

```
func (c *S3) PutBucketLifecycleConfiguration(input
*PutBucketLifecycleConfigurationInput) (*PutBucketLifecycleConfigurationOutput,
error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
LifecycleConfiguration	*BucketLifecycleConfiguration	封装生命周期的数组，最多可包含1000条规则	否

代码示例

```
func PutBucketLifecycleConfiguration(svc *s3.S3, bucket string) {
    // 设置匹配指定前缀的对象一天后过期
    rule := &s3.LifecycleRule{
        ID:     aws.String("expireAfterOneDay"),
        Status: aws.String("Enabled"),
        Filter: &s3.LifecycleRuleFilter{
            Prefix: aws.String("expireAfterOneDay/"),
        },
        Expiration: &s3.LifecycleExpiration{
            Days: aws.Int64(1),
        },
    }
    input := &s3.PutBucketLifecycleConfigurationInput{
        Bucket: aws.String(bucket),
        LifecycleConfiguration: &s3.BucketLifecycleConfiguration{
            Rules: []*s3.LifecycleRule{rule},
        },
    }
    output, err := svc.PutBucketLifecycleConfiguration(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

删除生命周期规则

功能介绍

删除指定桶的生命周期规则。

方法原型

```
func (c *S3) DeleteBucketLifecycle(input *DeleteBucketLifecycleInput)
(*DeleteBucketLifecycleOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

代码示例

```
func DeleteBucketLifecycle(svc *s3.S3, bucket string) {
    input := &s3.DeleteBucketLifecycleInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.DeleteBucketLifecycle(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

对象基础操作

获取对象

功能介绍

下载对象。

方法原型

```
func (c *S3) GetObject(input *GetObjectInput) (*GetObjectOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `Key`	是
VersionId	*string	用于获取对象的特定版本	否

输出结果

参数	类型	说明
AcceptRanges	*string	指出一段字节的范围
Body	io.ReadClosed	对象数据
ContentLength	*int64	以字节为单位的对象数据长度
ContentType	*string	描述对象数据格式的 MIME 类型
ETag	*string	对象的 `ETag`
LastModified	*time.Time	最后修改对象的时间
TagCount	*int64	对象上标签的数量(如果有)
VersionId	*string	对象版本

代码示例

```
func GetObject(svc *s3.S3, bucket string) {
    input := &s3.GetObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    output, err := svc.GetObject(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

上传对象

功能介绍

上传对象。单次上传最大不能超过5GB，超过5GB的文件需要使用分段上传操作。

方法原型

```
func (c *S3) PutObject(input *PutObjectInput) (*PutObjectOutput, error)
```

输入参数

参数	类型	说明	是否必要
ACL	*string	配置上传对象的预定义 ACL 信息，如`private`、`public-read`、`public-read-write`等。不传该参数时默认为`private`	否
Body	io.ReadSeeker	对象的数据	是
Bucket	*string	执行本操作的桶名称	是
Key	*string	对象的名称，如需上传到`/folder`目录下，只需设置`Key`为`/folder/{Key}`即可	是
Tagging	*string	对象的标签信息，必须是 URL 请求参数的形式。如`Key1=Value1`	否

输出结果

参数	类型	说明
ETag	*string	上传对象后对应的`Entity Tag`
VersionId	*string	上传对象后对应的版本号

代码示例

```
func PutObject(svc *s3.S3, bucket string) {
    input := &s3.PutObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
        Body:   bytes.NewReader([]byte("exampleFile")),
    }
    output, err := svc.PutObject(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取对象列表

功能介绍

列出指定桶中的全部对象，该操作最多返回1000个对象信息，可以通过设置过滤条件来列出桶中符合特定条件的对象。

方法原型

```
func (c *S3) ListObjects(input *ListObjectsInput) (*ListObjectsOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Delimiter	*string	一个分隔符，用于分组键	否
Maker	*string	指定开始列出对象的分隔 `key`（按字典序）	否
MaxKeys	*int64	最大返回数量，默认和最大值均为1000	否
Prefix	*string	返回以指定前缀开头的 `key`	否

输出结果

属性名	类型	说明
CommonPrefixes	[]*CommonPrefix	当请求中设置了 `Delimiter` 和 `Prefix` 属性时，所有包含指定 `Prefix` 且第一次出现 `Delimiter` 字符的对象 `key` 作为一组
Contents	[]*Object	对象的元数据
Delimiter	*string	与请求中设置的 `Delimiter` 相同
IsTruncated	*bool	表示是否返回满足搜索条件的所有结果的标志
Maker	*string	与请求中设置的 `Maker` 相同
MaxKeys	*int64	返回结果的对象数量的最大值
Name	*string	执行本操作的桶名称
NextMaker	*string	当响应被截断时（`IsTruncated` 为 true），可以在下次查询请求时用该字段作为 `Marker` 来获取下一组对象
Prefix	*string	与请求中设置的 `Prefix` 一致

代码示例

```go
func ListObjects(svc *s3.S3, bucket string) {
    input := &s3.ListObjectsInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.ListObjects(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

删除对象

功能介绍

`DeleteObject` 操作用于删除一个对象。对于开启了版本控制的桶执行该操作时，若指定版本号，则删除该版本的对象; 若未指定版本号，则将保留对象的当前版本并插入删除标记（Delete Marker），`DeleteObjectOutput` 中 `DeleteMarker` 将为 true，后续若执行 `GetObject` 操作将检测到该标记并返回`404`错误。

方法原型

```go
func (c *S3) DeleteObject(input *DeleteObjectInput) (*DeleteObjectOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	指定要删除的对象的版本号	否

输出结果

属性名	类型	说明
DeleteMarker	*bool	桶开启版本控制下，未指定版本号将得到该值为 true 的返回值
VersionId	*string	删除标记所在的版本号

代码示例

```
func DeleteObject(svc *s3.S3, bucket string) {
    input := &s3.DeleteObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    output, err := svc.DeleteObject(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

批量删除对象

功能介绍

批量删除多个对象，可以减少多次请求的重复花销。一次 `DeleteObjects` 操作最多可包含1000个 key 的删除请求。

方法原型

```
func (c *S3) DeleteObjects(input *DeleteObjectsInput) (*DeleteObjectsOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Delete	*Delete	封装了要删除的的对象和返回模式的属性	是

输出结果

参数	类型	说明
Deleted	[]*DeletedObject	被删除的对象的信息数组，每项都包含了一个被成功删除的对象的删除标记、`key` 和版本号等信息
Errors	[]*Error	删除失败的对象的信息数组，每项都包含了一个被成功删除的对象的信息和错误码

代码示例

```
func DeleteObjects(svc *s3.S3, bucket string) {
    input := &s3.DeleteObjectsInput{
        Bucket: aws.String(bucket),
        Delete: &s3.Delete{
            Objects: []*s3.ObjectIdentifier{
                {
                    Key: aws.String("exampleKey1"),
                },
                {
```

```
                    Key: aws.String("exampleKey2"),
                },
            },
        },
    }
    output, err := svc.DeleteObjects(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

复制对象

功能介绍

将已有的对象拷贝到其他位置。

方法原型

```
func (c *S3) CopyObject(input *CopyObjectInput) (*CopyObjectOutput, error)
```

输入参数

参数	类型	说明	是否必要
ACL	*string	预定义的 ACL 信息	否
Bucket	*string	目标桶的名称	是
CopySource	*string	URL 格式的原对象路径，可以用 URL 请求参数的形式指定版本号，如 `{bucketName}/{objectName}?versionId={versionId}`	否
Key	*string	目标对象的 key	是

输出结果

参数	类型	说明
CopyObjectResult	*CopyObjectResult	包含目标对象的 `ETag` 和最后修改时间等信息

代码示例

```
func CopyObject(svc *s3.S3, bucket string) {
    input := &s3.CopyObjectInput{
        Bucket:     aws.String(bucket),
        Key:        aws.String("exampleKey"),
        CopySource: aws.String("exampleBucket2/exampleKey"),
    }
    output, err := svc.CopyObject(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取对象元数据

功能介绍

获取指定对象的元数据信息。

方法原型

```
func (c *S3) HeadObject(input *HeadObjectInput) (*HeadObjectOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	用于获取对象的特定版本	否

输出结果

参数	类型	说明
ContentLength	*int64	以字节为单位的对象数据长度
ContentType	*string	描述对象数据格式的MIME类型
ETag	*string	对象的 `ETag`
LastModified	*time.Time	最后修改对象的时间
VersionId	*string	对象版本

代码示例

```
func HeadObject(svc *s3.S3, bucket string) {
    input := &s3.HeadObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    output, err := svc.HeadObject(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取对象标签

功能介绍

查询对象的标签信息。

方法原型

```
func (c *S3) GetObjectTagging(input *GetObjectTaggingInput)
(*GetObjectTaggingOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

输出结果

参数	类型	说明
TagSet	[]*Tag	对象标签信息数组

代码示例

```
func GetObjectTagging(svc *s3.S3, bucket string) {
    input := &s3.GetObjectTaggingInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    output, err := svc.GetObjectTagging(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

设置对象标签

功能介绍

为对象设置标签。

方法原型

```
func (c *S3) PutObjectTagging(input *PutObjectTaggingInput)
(*PutObjectTaggingOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的key	是
Tagging	*Tagging	封装标签键值对的数组	是
VersionId	*string	对象的版本号	否

代码示例

```
func PutObjectTagging(svc *s3.S3, bucket string) {
    tagging := &s3.Tagging{
        TagSet: []*s3.Tag{
            {
                Key:   aws.String("key1"),
                Value: aws.String("value1"),
            },
        },
    }
    input := &s3.PutObjectTaggingInput{
        Bucket:  aws.String(bucket),
        Key:     aws.String("exampleKey"),
        Tagging: tagging,
    }
    output, err := svc.PutObjectTagging(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

删除对象标签

功能介绍

删除指定对象的全部标签信息。删除时可以指定版本号参数来删除指定版本的对象的标签信息，不指定时默认操作于当前版本。

方法原型

```
func (c *S3) DeleteObjectTagging(input *DeleteObjectTaggingInput)
(*DeleteObjectTaggingOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

代码示例

```
func DeleteObjectTagging(svc *s3.S3, bucket string) {
    input := &s3.DeleteObjectTaggingInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    output, err := svc.DeleteObjectTagging(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取对象访问权限

功能介绍

获取指定对象的访问权限信息。

方法原型

```
func (c *S3) GetObjectAcl(input *GetObjectAclInput) (*GetObjectAclOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

输出结果

参数	类型	说明
Grants	[]*Grant	授权信息的数组，包含了每项授权和被授权人的信息
Owner	*Owner	对象所在桶的拥有者信息，包含了用户名和用户 id 等信息

代码示例

```
func GetObjectAcl(svc *s3.S3, bucket string) {
    input := &s3.GetObjectAclInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    output, err := svc.GetObjectAcl(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

设置对象访问权限

功能介绍

为指定对象设置访问权限。

方法原型

```
func (c *S3) PutObjectAcl(input *PutObjectAclInput) (*PutObjectAclOutput, error)
```

输入参数

参数	类型	说明	是否必要
ACL	*string	预定义的标准ACL信息，常用的有 `private` 私有读写、`public-read` 公共读、`public-read-write` 公共读写等	否
AccessControlPolicy	*AccessControlPolicy	配置该对象对于每个用户的 ACL 授权信息	否
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

`AccessControlPolicy` 的内部结构如下：

参数	类型	说明	是否必要
Grants	[]*Grant	授权信息	是
Owner	*Owner	所有者	是

`Grants` 的内部结构如下：

参数	类型	说明	是否必要
Grantee	*Grantee	被授权者	是
Permission	*string	授予权限，可选值为 `READ`，`READ_ACP`、`WRITE`、`WRITE_ACP` 和 `FULL_CONTROL`	是

`Owner` 的内部结构如下：

参数	类型	说明	是否必要
DisplayName	*string	所有者的用户名称	是
ID	*string	所有者的用户 ID	是

`Grantee` 的内部结构如下：

参数	类型	说明	是否必要
DisplayName	*string	用户名称	否
EmailAddress	*string	用户邮箱	否
Type	*string	用户类型，可选值为 `CanonicalUser`、`AmazonCustomerByEmail` 和 `Group`。`CanonicalUser` 则 `ID` 为必填项；`AmazonCustomerByEmail` 则 `EmailAddress` 为必填项，并且将会保存其指向到的 `CanonicalUser` 类型的用户；`Group` 则 `URI` 为必填项。	是
ID	*string	标准用户 ID	否
URI	*string	授权组的 URI	否

代码示例

```
func PutObjectAcl(svc *s3.S3, bucket string) {
    input := &s3.PutObjectAclInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
        ACL:    aws.String("private"),
    }
    output, err := svc.PutObjectAcl(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取对象版本信息

功能介绍

列出指定桶中的全部对象的版本信息，该操作最多返回1000个对象信息，可以通过设置过滤条件来列出桶中符合特定条件的对象。

方法原型

```
func (c *S3) ListObjectVersions(input *ListObjectVersionsInput)
(*ListObjectVersionsOutput, error)
```

输入参数

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Delimiter	*string	若设置了 `Prefix`，所有包含指定 `Prefix` 且第一次出现 `Delimiter` 字符的对象 `key` 作为一组。若未指定 `Prefix` 参数，按 `Delimiter` 对所有对象 `key` 进行分割	否
KeyMarker	*string	指示从哪里开始列出，`ZOS` 会在这个指定的键之后开始列出，即列出名称大于 `keyMarker` 的对象。标记可以是桶中的任何键。默认值为空字符串 ""。	否
MaxKeys	*int64	响应中返回的对象 `key` 的数量，最大值和默认值均为1000	否
Prefix	*string	返回的 `key` 的前缀，注意 `key` 与 `Prefix` 完全相同的对象不会返回。默认值为空字符串 ""。	否

输出结果

属性名	类型	说明
CommonPrefixes	[]*CommonPrefix	当请求中设置了 `Delimiter` 和 `Prefix` 属性时，所有包含指定 `Prefix` 且第一次出现 `Delimiter` 字符的对象 `key` 作为一组
DeleteMarkers	[]*DeleteMarkerEntry	对象删除标记数组，每一项包含了是否为最新版本，对象 `key`，最新修改时间，拥有者和版本号等信息
Delimiter	*string	与请求中设置的 `Delimiter` 相同
EncodingType	*string	返回对象 `key` 的字符编码类型
IsTruncated	*bool	表示是否返回满足搜索条件的所有结果的标志
KeyMarker	*string	与请求中设置的 `KeyMarker` 相同
MaxKeys	*int64	返回结果的对象数量的最大值
Name	*string	执行本操作的桶名称
NextKeyMaker	*string	当响应被截断时（`IsTruncated` 为 `true`），可以在下次查询请求时用该字段作为 `KeyMarker` 来获取下一部分的对象版本信息
NextVersionIdMarker	*string	本次查询返回的最后一个对象的版本号
Prefix	*string	与请求中设置的 `Prefix` 一致
Versions	[]*ObjectVersion	对象版本信息数组

代码示例

```
func ListObjectVersions(svc *s3.S3, bucket string) {
    input := &s3.ListObjectVersionsInput{
        Bucket: aws.String(bucket),
    }
    output, err := svc.ListObjectVersions(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

进阶操作

预签名下载

功能介绍

通过预签名链接下载对象。

使用`Presign`方法可以对请求进行预签名，生成预签名链接。任意用户在预签名链接的有效期内均可通过该链接完成`GetObject`请求。

预签名下载链接的使用方式可参考下面的指令，将文件名和链接替换为自己的参数即可。

```
curl -Lo your_local_file_name 'your_presign_url'
```

方法原型

```
// 用于构造 `GetObject` 请求
func (c *S3) GetObjectRequest(input *GetObjectInput) (req *request.Request,
output *GetObjectOutput)
// 用于预签名请求
func (r *Request) Presign(expire time.Duration) (string, error)
```

输入参数

构造`GetObject`请求所使用的输入参数请参考对象基础操作的“获取对象”小节。这里只描述预签名方法所需的输入参数。

参数	类型	说明	是否必要
expire	time.Duration	预签名链接的过期时间	是

输出结果

属性名	类型	说明
u	string	预签名链接

代码示例

```
func PresignGetObjectRequest(svc *s3.S3, bucket string) {
    input := &s3.GetObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    // 构造请求
    request, _ := svc.GetObjectRequest(input)
    // 设定预签名链接过期时间
    expire := 15 * time.Minute
    // 预签名请求
    output, err := request.Presign(expire)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
```

```
}
```

预签名上传

功能介绍

通过预签名链接上传对象。

使用`Presign`方法可以对请求进行预签名，生成预签名链接。任意用户在预签名链接的有效期内均可通过该链接完成`PutObject`请求。

预签名上传链接的使用方式可参考下面的指令，将文件名和链接替换为自己的参数即可。

```
curl -v -k -X PUT -T your_local_file_name 'your_presign_url'
```

方法原型

```
// 用于构造 `PutObject` 请求
func (c *S3) PutObjectRequest(input *PutObjectInput) (req *request.Request,
output *PutObjectOutput)
// 用于预签名请求
func (r *Request) Presign(expire time.Duration) (string, error)
```

输入参数

构造`PutObject`请求所使用的输入参数请参考对象基础操作的“上传对象”小节。这里只描述预签名方法所需的输入参数。

参数	类型	说明	是否必要
expire	time.Duration	预签名链接的过期时间	是

输出结果

属性名	类型	说明
u	string	预签名链接

代码示例

```
func PresignPutObjectRequest(svc *s3.S3, bucket string) {
    input := &s3.PutObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String("exampleKey"),
    }
    // 构造请求
    request, _ := svc.PutObjectRequest(input)
    // 设定预签名链接过期时间
    expire := 15 * time.Minute
    // 预签名请求
    output, err := request.Presign(expire)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
```

```
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

附录

错误码总表

请求时返回的错误码含义可参考下表。

HTTP状态码	错误码	说明
403	AccessDenied	访问被拒绝
400	BadDigest	指定的 Content-MD5 与接收到的不匹配
409	BucketAlreadyExists	指定的桶已存在
409	BucketNotEmpty	桶内非空
400	EntityTooSmall	上传对象小于允许的最小大小
400	EntityTooLarge	上传对象超过允许的最大大小
400	ExpiredToken	提供的令牌已过期
400	IncompleteBody	未提供 Content-Length HTTP 头中指定的字节数
400	IncorrectNumberOfFilesInPostRequest	POST 请求每个请求只允许上传一个文件
500	InternalError	内部错误，请重试
403	InvalidAccessKeyId	无效的访问密钥
400	InvalidArgument	无效的参数
400	InvalidBucketName	无效的桶名称
409	InvalidBucketState	请求与桶的当前状态不符
400	InvalidDigest	指定的 Content-MD5 无效
403	InvalidObjectState	请求与对象的当前状态不符
400	InvalidPart	无效的分段
400	InvalidPartOrder	无效的分段顺序
400	InvalidPolicyDocument	无效的策略参数
416	InvalidRange	无法的请求范围
403	InvalidSecurity	无效的秘密密钥
400	InvalidStorageClass	无效的存储类型
400	InvalidTargetBucketForLogging	无效的日志目标存储桶
400	InvalidToken	无效的令牌格式
400	InvalidURI	无法解析指定的URI
400	KeyTooLongError	对象名过长
400	MalformedACLError	错误的 ACL 格式
400	MalformedPOSTRequest	错误的 POST 请求体格式

HTTP状态码	错误码	说明
400	MalformedXML	错误的 XML 格式
400	MaxMessageLengthExceeded	请求信息过长
405	MethodNotAllowed	指定的方法不被允许对此资源进行操作
411	MissingContentLength	未提供 Content-Length HTTP 头
400	MissingRequestBodyError	缺少请求体
404	NoSuchBucket	指定的桶不存在
404	NoSuchBucketPolicy	指定的桶没有桶策略
404	NoSuchKey	指定的对象不存在
404	NoSuchLifecycleConfiguration	指定的桶没有生命周期配置
404	NoSuchUpload	指定的分段上传不存在
404	NoSuchVersion	请求中指定的版本号与现有版本不匹配
409	OperationAborted	操作被中止，请重试
307	Redirect	临时重定向
409	RestoreAlreadyInProgress	解冻进行中
400	RequestIsNotMultiPartContent	POST 请求体不符合 multipart/form-data 格式
400	RequestTimeout	请求超时
403	RequestTimeTooSkewed	请求时间与服务器的时间差值过大
403	SignatureDoesNotMatch	服务计算的请求签名与您提供的签名不匹配
400	TooManyBuckets	桶数量超额
400	TokenRefreshRequired	提供的令牌需要刷新
400	UnresolvableGrantByEmailAddress	提供的电子邮件地址与记录中的任何帐户都不匹配